The 2015
Fall
Conference of
the KIPS

2015년
추계학술발표대회
논문집

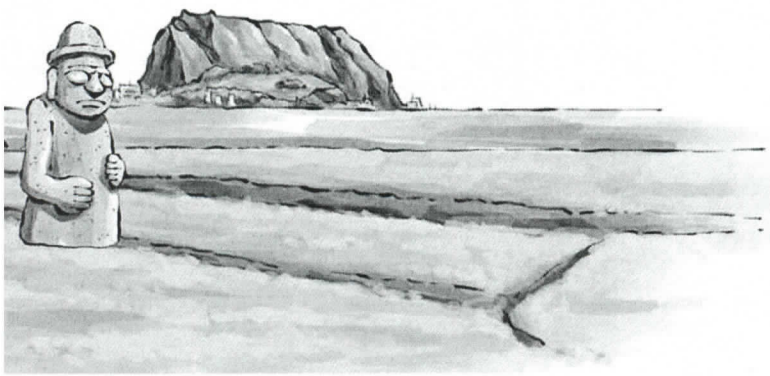
일 자 2015년 10월 30일(금) ~ 31일(토)

장 소 제주한라대학교

주 최 한국정보처리학회

주 관 제주한라대학교 정보기술교육원

협 찬 SAMSUNG 삼성SDS, LG히다찌, KCC정보통신, 한국게임과학고등학교 Copy Killer, NAVER, SJ정보통신, KWANG MYUNG D&C 광명 D&C, MarkAny, BIT 비트컴퓨터, 올포랜드, 중앙정보처리학원 Since 1969 중앙정보기술인재개발원, TRUE NETWORKS CONNECT THE WORLD, kpc 한국생산성본부



사단법인 한국정보처리학회
KIPS Korea Information Processing Society

코드 분석을 위한 JDT 기반 정적 분석기 개발

박민규*, 변은영*, 한정화*, 김영철*, 문소영**
*홍익대학교 컴퓨터정보통신공학과
**홍익대학교 컴퓨터정보통신공학과 소프트웨어공학연구실
hello0922@gmail.com*, uwuwu@naver.com*, hjhhahahoho@naver.com*,
bob@hongik.ac.kr*, msy@selab.hongik.ac.kr**

Development of JDT Based Static Analyzer for Code Analysis

Min-Gyu Park*, Eun-Young Byun*,
Jeong-Wha Han*, Robert Youngchul Kim*, So-Young Moon**
*Dept of Computer Information Communication, Hongik University
**SE Lab., Dept of Computer Information Communication, Hongik University

요 약

오늘날 소프트웨어의 크기는 계속 증가하고 있는데 반해 IT 벤처/중소 업체의 경우 요구사항 및 설계 문서가 없는 경우가 빈번하다. 이러한 시스템의 경우 코드를 이해하여 수정이나 유지보수를 하는데 많은 시간과 비용이 투자된다. 또한 벤처/중소 업체에서 역공학 도구, 테스트 프로세스 등을 도입하여 소프트웨어 품질 향상을 시키기는 현실적으로 비용 면에서 어려움이 있다. 본 연구는 내재된 코드의 오류를 찾기 위해 JDT 기반 정적 분석기를 제안한다. 제안한 분석기의 설계 구조 및 구현으로 개발자 주도 코드 분석을 통해 코드 품질을 향상 시킬 것이다. 또한 요구사항과 코드의 불일치에 대한 가시화를 통해 소프트웨어의 유지보수성을 향상 시킬 것이다.

1. 서론

소프트웨어 개발에 있어 개발자 개인 의존도가 높으며 요구사항 및 설계 문서가 없는 경우기 빈번하다. 또한 개발자의 잦은 이직으로 소프트웨어의 수정 및 확장에 어려움을 겪는 벤처/중소 업체들이 있다. 그러나 만약 이 경우 코드와 일치하는 설계문서가 있다면 코드를 이해하고 수정하는데 도움을 줄 수 있다. 그래서 역공학 도구를 도입하여 분석하는 회사들도 있지만, 기존 역공학 도구들은 텍스트 및 그래픽 보고서 출력 등의 기능을 포함하고 코드를 문서로 보여주는 기능만 갖고 있어 보완이 필요하다. 또한 IT 벤처/중소기업에서 코드의 품질을 높이기 위해 역공학 도구나 SW 개발 프로세스 및 방법론, 테스트 프로세스 및 테스팅을 사용하기는 비용 부담이 크다. 본 논문에서는 코드 복잡도, 코딩 규칙 준수 등을 적용하여 모듈 간 복잡도를 표현하여 코드를 보여주는 JDT 기반 맞춤형 코드 정적 분석 도구를 제안한다. 이는 첫째, 기존 소프트웨어의 비가시성을 제거하고 소프트웨어의 아키텍처 개선에 도움을 준다. 둘째, 고가의 상용 소프트웨어를 사용하 수 없는 IT 벤처/중소기업의 소프트웨어에 대한 체계적인 품질관리를 가능하도록 한다. 셋째, 개발 문서와 코드의 불일치 해소를 통해 소프트웨어의 유지보수성을 향상 시킨다.

본 논문 구성은 다음과 같다. 2장에서는 파서(Parser)와 가시화에 대해 언급하고 기존 정적 분석기와 비교를 한다. 3장에서는 정적 분석기 구성과 개발 방법에 대해 설명한다. 4장에서는 사례 연구로 본 논문에서 개발한 JDT 기반 정적 분석기를 사용하여 안드로이드 앱에 대한 정적 분석을 한다. 마지막으로 5장에서는 결론 및 향후 연구에 대해 언급한다.

2. 관련연구

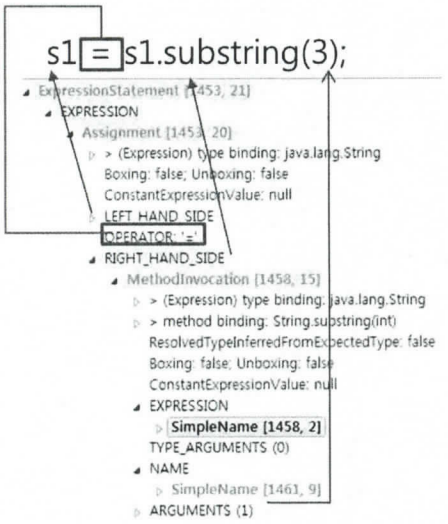
2.1 JDT를 위한 추상 구문 트리

파서(Parser)는 스캐너로부터 토큰으로 나누어진 원시 프로그램을 받아서 프로그램의 구조를 결정하는 구문 분석(syntax analysis)을 수행한다. 이는 자연어의 문장에서 문법 분석을 하는 것과 비슷하다. 구문 분석은 프로그램의 구조적 요소와 그들의 관계를 경정한다. 구문 분석의 결과는 구문 트리(Syntax Tree) 또는 추상 구문 트리(Abstract Syntax Tree, AST)로 나타낸다[1]. AST에서는 변수 선언문, 조건문, 반복문 등이 모두 Statement로 분류된다. 그리고 for문은 ForStatement라는 하나의 노드로 표현된다.

JDT(Java Development Tool)[2]는 이클립스에서 플러그인으로 제공된다. 자바 파서와 AST를 포함하고, 이를 통해 코드를 분석한다. JDT의 자바 모델은 트리형태로 내부의 요소들을 볼 수 있도록 한다. 내부 노드는 크게 package, compliation units, binary classes, types, methods, fields와 같은 것들로 구성된다. AST는 각 언어마다 제공되는데,

* 이 연구는 2015년 한국연구재단과 한국여성과학기술인지원센터의 지원을 받아 연구되었습니다.

자바를 위한 AST가 JDT이다. JDT는 코드를 분석해서 그 내용을 트리 구조로 뽑아준다. (그림 1)은 JDT의 ASTView를 통해 본 트리 구조이다. s1=s1.subasting(3)에서 s1은 LEFT_HAND_SIDE에 속하고, s1.substring(3)은 RIGHT_HAND_SIDE이다. 그리고 대입연산자 '='는 OPETATOR로 분류된다. 이 형태로 모든 코드가 트리 형태로 분석이 된다.



(그림 1) ASTView

2.2 기존 정적 분석 도구

아래 <표 1>은 기존 정적 분석 도구를 비교하여 사용 가능 언어, 구문 분석 범위, 가시화 기능, 단점, 커스터마이징 가능에 따라 정리하였다. Source Navigator[4]는 소스 코드 분석 도구로 클래스, 함수, 변수등에 대해 트리 구조로 보여준다. 사용 가능한 언어로는 Fortran, Cobol, Java, C++ 등이 있으며 가시화 기능으로는 코드 트리 브라우징만을 제공해준다. SonarQube[5]는 Java, C/C++, C#외에도 다수의 언어를 제공하며, 클래스, 함수, 변수, 메서드 등으로 구문 분석이 가능하다. 가시화 기능으로는 코딩 설계 및 구조등에 대해 가능하고, 단점으로는 자바는 무료지만 몇몇 다른 플러그인은 유료로 제공되고, 커스터마이징이 불가능하다. SourceInsight[6]는 C/C++, Java, C# 등의 언어를 지원하고, 클래스, 함수, 변수 등의 구문을 분석한다. 가시화 기능으로는 호출 그래프, 클래스 트리 다이어그램을 제공한다. 다양한 편집 기능이 제공되나 출력되는 가시화 종류가 적다. 이 도구도 커스터마이징은 불가능하다.

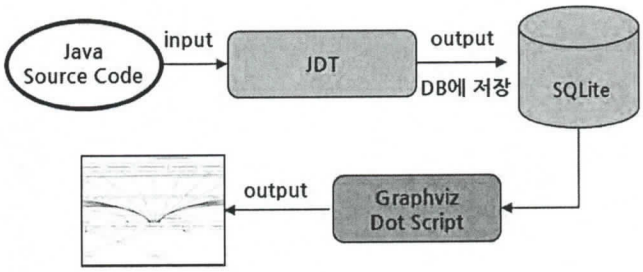
<표 1> 기존 정적 분석 도구 비교

도구	사용 가능 언어 (대표)	구문 분석 범위	가시화 기능	단점	커스터마이징
Source Navigator (오픈소스)	Fortran, Cobol, Java, C++	class, function, variable, method, structures, 등	·코드 트리 브라우징	·가시화 기능이 약함. ·다이어그램 제공되지 않음.	X
SonarQube (오픈/상용)	Java, C/C++, C, C#	class, function, variable, method, structures, 등	·코딩 설계 및 구조등의 가시화	·자바는 무료, 몇몇의 다른 플러그인은 유료 제공. ·가시화 기능이 있지만, 다양한 설계 다이어그램 가시화는 제공하지 않음.	X
SourceInsight (상용)	C/C++, C#, Java	functions, methods, global variables, structures, classes 및 다른 타입의 기호(symbol)	·호출 그래프 ·클래스 트리 다이어그램	·다양한 편집 기능이 제공되나, 출력되는 가시화 종류가 적음.	X

3. JDT 기반 정적 분석기

3.1 JDT 기반 정적 분석기 구조

Java 소스 코드를 JDT로 분석하여 그 내용을 DB로 저장하고 DB에서 데이터를 추출하여 DOT 스크립트를 작성하고 Graphviz 도구를 사용해 원하는 다이어그램을 출력한다.



(그림 2) JDT 기반 정적 분석기 구조

DOT 스크립트는 작성자에 따라 원하는 다양한 형태의 그림을 그릴 수 있어 확장이 용이하다. JDT 기반 코드 분석

은 기존의 도구들과 다르게 우리가 원하는 형태의 다이어그램을 추출이 가능하다. 그 이유는 세부적인 부분까지 모두 분석해서 데이터베이스화 하기 때문에 코드에 있는 모든 내용을 바탕으로 원하는 산출물 결과를 얻을 수 있다.

3.2 JDT 기반 코드 분석 데이터베이스화

```
package test;
import java.util.Scanner;
public class ArrayAccess {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int intArray[];
        intArray = new int[5];
        int max=0;
        System.out.println("정수 5개를 입력하세요. ");
        for(int i=0; i<5; i++) {
            intArray[i] = scanner.nextInt(); // 입력받은 정수를 배열에 저장
            if(intArray[i] > max)
                max = intArray[i];
        }
        System.out.print("가장 큰 수는 " + max + "입니다.");

        scanner.close();
    }
}
```

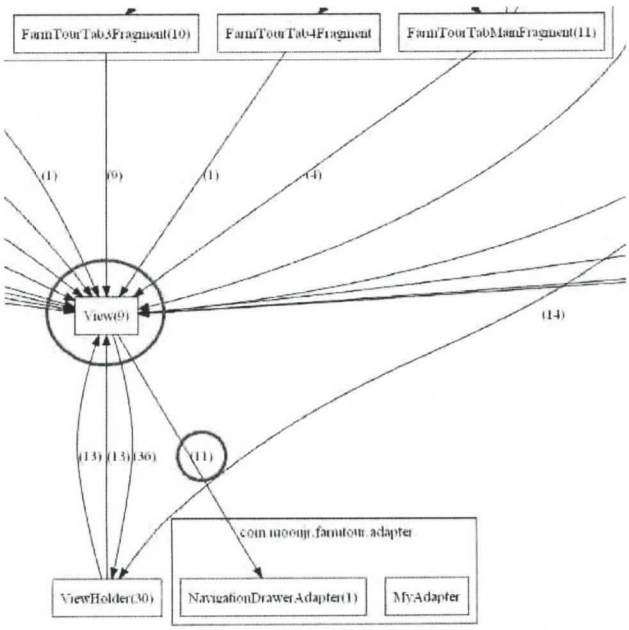
```
BODY
  Block [110, 369]
    STATEMENTS (8)
      VariableDeclarationStatement [115, 41]
      VariableDeclarationStatement [162, 15]
      ExpressionStatement [182, 22]
      VariableDeclarationStatement [209, 10]
      ExpressionStatement [223, 36]
      ForStatement [265, 138]
        INITIALIZERS (1)
        EXPRESSION
        UPDATERS (1)
        BODY
          Block [288, 115]
            STATEMENTS (2)
              ExpressionStatement [294, 32]
              IfStatement [350, 47]
                EXPRESSION
                  InfixExpression [353, 17]
                    > (Expression) type binding: boolean
                    Boxing: false; Unboxing: false
                    ConstantExpressionValue: null
                    LEFT_OPERAND
                    OPERATOR: '>'
                    RIGHT_OPERAND
                    EXTENDED_OPERANDS (0)
                THEN_STATEMENT
                  ExpressionStatement [379, 18]
                    EXPRESSION
                      Assignment [379, 17]
                        > (Expression) type binding: int
                        Boxing: false; Unboxing: false
                        ConstantExpressionValue: null
                        LEFT_HAND_SIDE
                        OPERATOR: '='
                        RIGHT_HAND_SIDE
                ELSE_STATEMENT: null
      ExpressionStatement [407, 44]
      ExpressionStatement [459, 16]
```

(그림 3) 배열 예제와 JDT 분석 구조

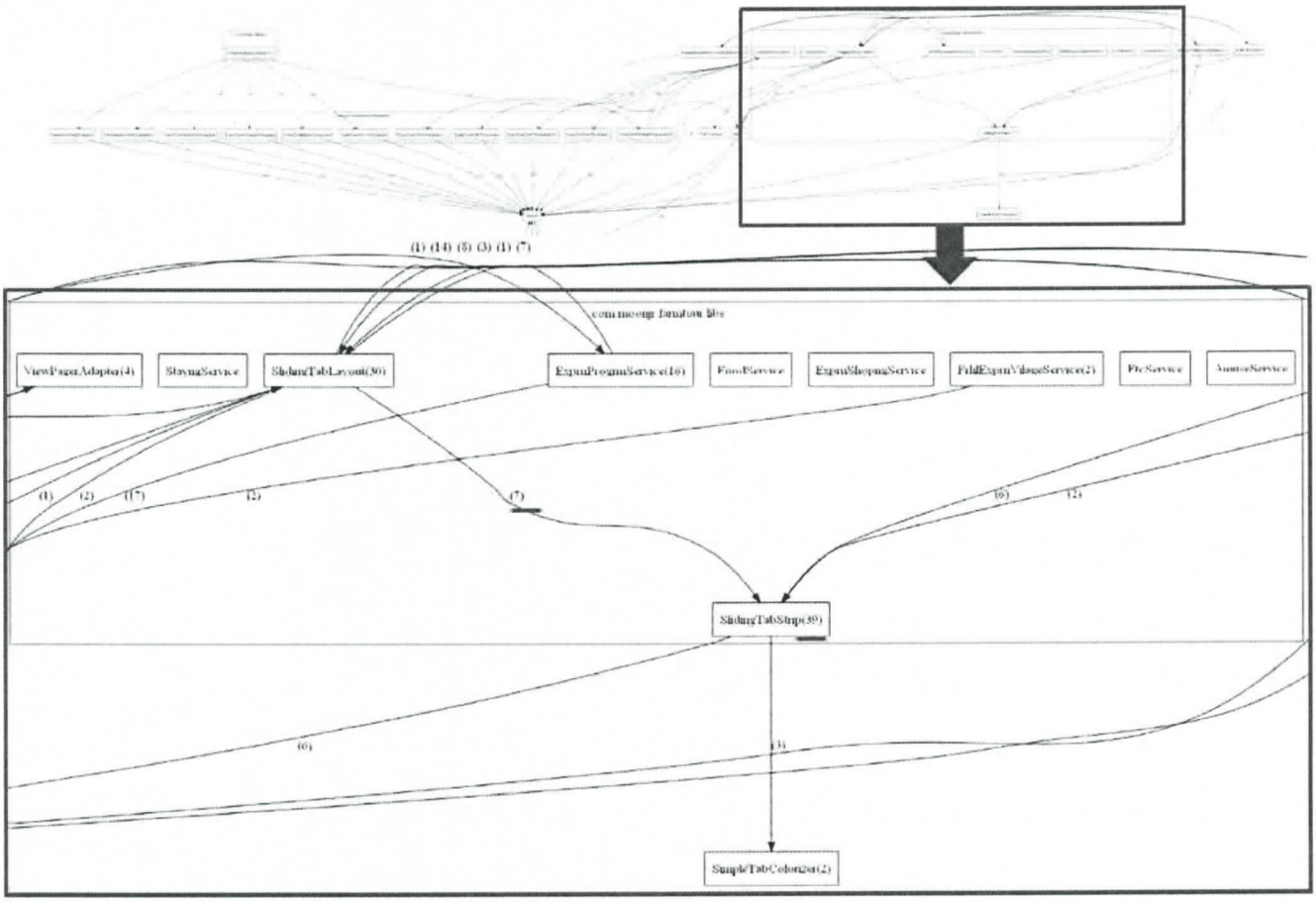
(그림 3)의 상단에 있는 소스는 배열에 5개의 값을 입력받아 가장 큰 수를 구하는 자바 프로그램이다. 위 소스 코드는 for문을 예로 설명한다. for문은 AST에서 ForStatement 노드로 분류되고, INITIALIZERS, EXPRESSION, UPDATERS를 포함한다. "{}"는 Block으로 분류하여 블록 내에 있는 문장들은 STATEMENTS 노드로 분류된다. "intArray[i]=scanner.nextInt()" 문장은 ExpressionStatement로 크게 분류되고, if문은 IfStatement로 표현된다. if(intArray[i] > max)에서 '>'는 OPERATOR가 되고 intArray[]는 LEFT_OPERAND로 분류, max는 RIGHT_OPERANDS가 된다. 그리고 max=intArray[i] 문장은 THEN_STATEMENT로 크게 분류된다. 그리고 다시 ExpressionStatement → EXPRESSION → Assignment 로 노드가 결정된다. Assignment 노드에서 '='는 OPERATOR가 되고, max는 LEFT_OPERAND, intArray[i]는 RIGHT_OPERAND로 분류된다. 이렇게 분류되는 데이터들을 클래스, 변수, 메서드, 문장 등등 별로 저장하여 데이터베이스화 한다.

4. 사례연구

본 장에서는 안드로이드 앱을 대상으로 코드 분석을 수행한다. 자바 소스를 읽어 자바 파서를 통해 분석하고 JDT 기반 AST로 변경된 내용을 데이터베이스화 하고 그 내용을 DOT 스크립트로 작성한 후 가시화한 결과가 (그림 4)와 (그림 5)이다. (그림 4)의 View 클래스는 View 클래스 내부에서 메서드를 호출하는 횟수가 9번이고, View에서 NavigationDrawerAdapter 클래스의 메서드를 사용하는 횟수는 11회 이다. (그림 5)는 가시화된 큰 화면을 춤아웃하여 일부를 보여준다. SlidingTabLayout 클래스에서 SlidningTabStrip 클래스를 7번 참조하고, SlidingTabStrip은 내부사용이 39회 임을 나타낸다.



(그림 4) JDT 기반 정적 분석기를 통한 코드 가시화



(그림 5) 안드로이드 앱 가시화

본 논문에서 개발한 JDT 기반 정적 분석기는 다음과 같은 이점이 있다. 첫째, 개발자가 개발 중인 프로젝트에 대해 가시화하여 소스 코드의 복잡도를 체크하고, 클래스와 클래스 간의 메서드 사용에 대한 연관 관계, 객체 생성 등을 체크한다. 또한 다른 클래스와 상관없는 클래스를 찾아 낼 수 있는 장점이 있다. 이는 개발자 스스로가 코드의 품질을 향상시킬 수 있기 때문에 개발하면서 지속적인 코드 품질 관리가 가능하다는 것이다. 둘째, 개발자가 이직을 하고 완료된 프로젝트의 경우에 개발 문서가 존재하지 않는다면 새로운 개발자는 난감한 상황을 피할 수 없다. 이 경우 본 연구에서 개발한 정적 분석기를 사용하면 클래스 간의 연관 관계 분석이 가능하고 유지보수에 도움을 줄 수 있다. 셋째, IT 벤처/중소기업에서 고가의 비용으로 품질관리를 할 수 없는 것을 회사 내부의 아키텍처를 구성하여 관리할 수 있기 때문에 비용 효과를 얻을 수 있다. 앞서 언급한 장점 외에도 학교에서 학생들에게 본 연구에서 개발한 JDT 기반 정적 분석기를 사용하여 학생들에게 소프트웨어 품질관리에 대한 교육을 하고 학생들도 자신이 개발한 프로그램을 가시화하여 분석하고 수정하면서 품질 관리를 한다면 기업의 QA 담당으로 취업도 가능하다.

5. 결론 및 향후 연구

본 논문에서는 JDT 기반 정적 분석기를 개발했고, 개발한 도구를 통해 소프트웨어 가시화를 시도했다. 기존 도구와의 차별성과 기능성을 위해 단순히 코드 가시화만을 수행한 것이 아니라 특정한 아키텍처(스템프 결합도)를 적용해 안드로이드 앱의 코드를 분석했다. 그 결과로 코드의 복잡도를 한 눈에 볼 수 있는 다이어그램을 생산했고, 코드 분석에 많은 도움이 되는 것을 알았다. 향후에는 더 다양한 다이어그램을 생성하여 소프트웨어 개발, 유지보수, 품질관리에 도움을 주는 도구로 발전시키고자 한다. 또한 동적분석 방법도 병행 연구하여 소스코드 분석 기능을 극대화 시키고자 한다.

참고문헌

[1] Kenneth C. Louden, "Compiler Construction Principles and Practice (1st ed.)", Course Technology, 1997

[2] JDT, http://www.eclipse.org/jdt

[3] Kenneth C. Louden, "Compiler Construction Principles and Practice (1st ed.)", Course Technology, 1997

[4] SrouceNavigator, http://sourcenav.sourceforge.net

[5] SonarQube, http://www.sonarqube.org

[6] SourceInsight, http://www.sourceinsight.com

2015년 추계학술발표대회 논문집 제22권 제2호

발 행 일 : 서기 2015년 10월 21일 인쇄
서기 2015년 10월 28일 발행

발 행 인 : 박 두 순

발 행 처 : 사단법인 한국정보처리학회
KIPS Korea Information Processing Society

04376 서울시 용산구 한강대로 109, 1002호(한강로 2가 용성비즈텔)
TEL : (02) 2077-1414(代) FAX : (02) 2077-1472
http://www.kips.or.kr E-mail : kips@kips.or.kr

인 쇄 처 : (주)이환디앤비
((02) 2254-4301(代), E-mail : ewhan@ewhan.com)